데이터통신 과제

— hw04 —

제 출 일	2016. 04. 18.
분 반	01분반
담당교수	김상하 교수님
학 과	컴퓨터공학과
학 번	201302450
조 장	석 지 훈
조 원	이 상 현
	임 대 영
	정 윤 수

1. 실습개요

가. 실습 일시 및 장소

1) 실습 일시 : 2017. 04. 15. 10:00 ~ 2017. 04. 16. 24:00

2) 실습 장소 : 충남대학교 학생생활관 8동

나. 실습 목적

이번 실습의 목적은 저번에 과제로 하였던 하나의 컴퓨터 내에서 프로세스 간의 통신에서 더 나아가 서로 다른 두개의 컴퓨터에서 서로 통신을 하는 것을 목적으로 하고 있다.

Packet Driver를 이용해 네트워크로 연결된 두 PC사이에 채팅이나 파일을 전송한다. 이 과정들은 Thread를 이용해 구현되기 때문에 동시처리가 가능하다. 또한, 메시지나 파일의 크기의 제한 없이 전송될 수 있도록 Data Fragmentation기능을 구현한다.

다. 실습 시나리오

1) 두 대의 PC에서 각각 프로그램을 실행

2) 두 대의 PC는 네트워크로 연결

3) PC 1의 프로그램에서 전송 버튼을 클릭

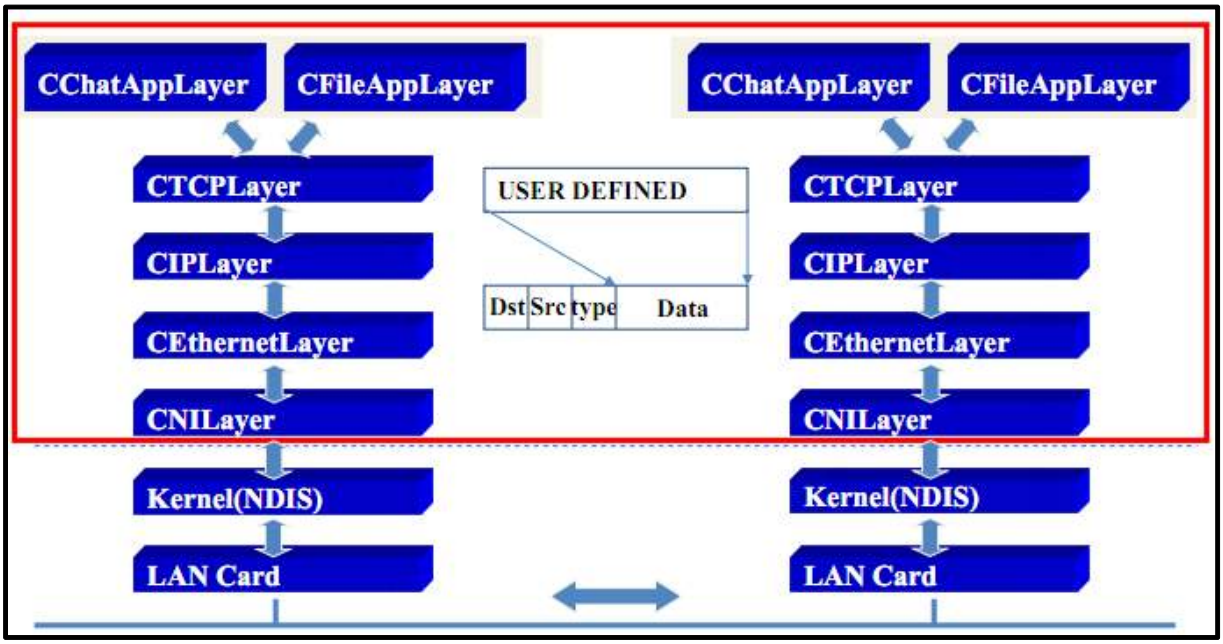
- 레이어 아키텍쳐에 의해서 만들어진 Ethernet frame이 네트워크로 전송됨

4) PC 2의 프로그램에서 Ethernet frame을 수신

- PC 1로부터 보내진 패킷이라면, Dialog 레이어까지 전달
- 전달된 결과를 화면에 팝업창으로 출력

2. 프로토콜 스택

가. 구조 설명(프로토콜의 역할 등)



1) IPCAppDlg : 이 계층은 프로그램의 전체적인 UI를 담당한다. 주로 송신에서는 계층을 만들고 연결하는 작업을 시작하며 ChatAppLayer에 입력받은 데이터를 전송을 한다. 수신에서는 마지막 패킷을 받을 때까지 수신과정을 반복하여 하나의 메시지로 합치고 전달받은 메시지를 화면에 출력한다.

2) ChatAppLayer : 이 계층은 송신자가 입력한 메시지 패킷을 처리하는 과정을 진행하며 응용 계층이다. 송신에서는 ChatAppLayer의 header에 data(메시지)를 붙여서 하위계층인 TCPLayer에게 전체를 전달하고 다음 송신과정을 진행한다. 수신에서는 TCPLayer로부터 받은 데이터 중에서 ChatAppLayer의 header를 분리한다. 이후 상위계층인 IPCAppDlg로 메시지를 전달하여 프로그램의 화면에 전달받은 메시지가 출력되도록 한다.

3) FileAppLayer : 이 계층은 송신자가 지정을 한 파일을 처리를 하여 전송을 과정을 담당을 하는 응용계층이다. 송신에서는 FileLayer의 header에 data(파일 데이터)를 붙여서 하위계층인 TCPLayer로 전송을 한다. 이 때 전송을 하고자 하는 파일의 크기가 1472byte 보다 크다면 단편화를 하여서 여러 개의 Packet으로 나누어서 전송을 한다. 수신에서는 TCPLayer로부터 전해 받은 Packet들의 data들을 자신의 원하는 경로에 새로운 파일을 생성을 하여 저장을 한다. 이 때 크기가 큰 파일은 여러 개의 Packet으로 나누어져서 FileLayer 계층으로 들어올 수 있다. FileLayer계층에서는 Packet의 header부분의 정보를 확인을 하여 처음 중간 마지막 부분의 따라서 다르게 처리를 한다.

4) TCPLayer : 이 계층은 송신자가 입력한 데이터를 처리하는 과정을 진행하며 전송 계층이다. 송신에서는 TCPLayer의 header에 data(메시지)를 붙여서 하위계층인 IPLayer에게 전체를 전달하고 다음 송신과정을 진행한다. 수신에서는 IPLayer로부터 받은 데이터 중에서 TCPLayer의 header를 분리한다. 이후 상위계층인 ChatAppLayer로 데이터를 전달을 한다. 특히 이 계층에서는 포트번호를 저장하는 필드를 따로 두어 송신에서는 port번호를 포함하여 전송하고, 수신에서는 port번호를 확인하여 패킷에 메시지가 들었는지 파일이 들었는지 구별한다. 만약 port번호가 0x2080이면 패킷에 메시지가 들어있으므로 헤더를 제외한 데이터를 ChatAppLayer로 보내고, port번호가 0x2090이면 파일을 전송한 것이므로 FileLayer로 보낸다.

5) IPLayer : 이 계층은 송신자가 입력한 데이터를 처리하는 과정을 진행하며 전송 계층이다. 송신에서는 IPLayer의 header에 data(메시지)를 붙여서 하위계층인 EthernetLayer에게 전체를 전달하고 다음 송신과정을 진행한다. 수신에서는 EthernetLayer로부터 받은 데이터 중에서 IPLayer의 header를 분리한다. 이후 상위계층인 TCP로 데이터를 전달을 한다.

6) EthernetLayer : 이 계층은 호스트간에 통신을 담당하는 계층으로 데이터 링크 계층이다. 송신에서는 IPLayer로부터 전달 받은 정보를 NILayer의 data(EthernetLayer header구조체의 enet_data필드)에 넣어서 하위계층인 NILayer에게 전체를 전송하고 다음 송신과정을 진행한다. 수신에서는 NILayer로부터 받은 데이터 중에서 Ethernet header를 분리한 Ethernet data부분(enet_data필드)를 상위계층인 IPLayer로 전달한다.

7) NILayer : 이 계층은 데이터를 패킷단위로 나누어서 네트워크로 연결된 다른 PC에 전송을 한다. 또한 수신에서는 연결된 네트워크로부터 받은 패킷들을 상위계층으로 보내준다.

3. 구현 설명

가. 전송

1_1 테스트패킷 전송) IPCAppDlg::OnBnClickedTestPacket()

```
void CIPCAppDlg::OnBnClickedTestPacket()
{
    CString MsgHeader;

    MsgHeader.Format("Send >> ");
    m_stMessage = "Test Packet";
    this->mp_UnderLayer->Send((unsigned char*) (LPCTSTR) m_stMessage, m_stMessage.GetLength(), CHAT_TYPE, CHAT_REC);
}
```

"테스트패킷" button을 클릭하면 호출되는 메소드로, 메시지나 파일이 아닌 테스트패킷만을 전송한다. 테스트패킷은 단순히 메시지에 "Test Packet"이라고 작성한 후, port를 CHAT_TYPE인 0x2080으로 설정한다. 이 후 하위 계층의 send()함수를 연속적으로 호출을 하여 입력을 받은 데이터를 전송을 한다.

1_2 메시지 전송) IPCAppDlg::OnSendMessage()

```
void CIPCAppDlg::OnSendMessage()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

    if (!m_stMessage.IsEmpty())
    {
        SendData();
        m_stMessage = "";

        (CEdit*) GetDlgItem(IDC_EDIT_MSG)->SetFocus();

    }

    UpdateData(FALSE);
}
```

"전송" button을 클릭하면 호출되는 메소드로, UpdateData(TRUE)를 이용하여 Main dialog에 적혀있는 글자를 클래스의 필드에 저장한다. 여기서 필드에 저장되는 값은 전송할 메시지로 m_stMessage에 저장된다. 이 후 전송할 메시지가 입력되었는지 확인하고, 메시지를 입력하였다면 실질적으로 메시지를 전송하는 AppDlg클래스의 SendData()를 호출한다.

1_3 파일 전송) OnButtonFileserch(), OnButtonFiletrans(), FileThread()

목차 바(Thread 사용), 사(File 전송)에서 설명하였음.

2) IPCAppDlg::SendData()

패킷으로 메시지를 전송할 때 IPCAppDlg::OnSendMessage()에서 호출되는 메소드로, 실질적인 메시지 전송을 담당한다. 먼저 전송하는 쪽 프로그램의 dialog에서 기본적으로 출력할 문자열 "Send >>"을 m_ListChat에 추가한다.

```
CString MsgHeader;
MsgHeader.Format("Send >> ");
int index = m_ListChat.AddString(MsgHeader);
```

그리고 m_stMessage를 통해 메시지의 길이를 확인하며, 길이가 30보다 작으면 바로 m_ListChat에 추가하고, 30보다 크면 m_ListChat에 잘라서 추가해준다. 참고로 m_ListChat은 수신자가 아닌 송신자의 dialog에 띄울 Chat의 모음이다.

```
if (m_stMessage.GetLength() > 30){//printing 30 Character(line) my display
    while (1){
        int index = m_ListChat.AddString(m_stMessage.Mid(len, 30));
        m_ListChat.SetCurSel(index);
        len += 30;
        if (m_stMessage.GetLength() - len < 30){
            index = m_ListChat.AddString(m_stMessage.Mid(len, m_stMessage.GetLength() - len));
            m_ListChat.SetCurSel(index);
            break;
        }
    }
}
else{
    int index = m_ListChat.AddString(m_stMessage);
    m_ListChat.SetCurSel(index);
}
```

다음으로는 위와 마찬가지로 일단 m_stMessage를 통해 메시지의 길이를 확인하고, 길이가 하나의 패킷에서 최대로 data를 저장할 수 있는 size인 APP_DATA_SIZE보다 크지 않으면 하위계층의 Send()를 이용하여 메시지 전송을 진행한다. 반대로 APP_DATA_SIZE보다 메시지의 길이가 길면 APP_DATA_SIZE만큼씩 메시지를 잘라서 여러 개의 패킷으로 나누어 전송한다. 여기서 Send의 parameter로 넘겨주는 CHAT_TYPE은 0x2080으로 TCPLayer에서 수신한 패킷이 file인지 message인지 구분하는데 사용된다.

```
if (m_stMessage.GetLength() > APP_DATA_SIZE){//bigger 경과 시간 1ms 이하
    while (1){
        if(len == 0){
            this->mp_UnderLayer->Send((unsigned char*) (LPCTSTR) m_stMessage.Mid(len, APP_DATA_SIZE), APP_DATA_SIZE, CHAT_TYPE, CHAT_REC);
        len += APP_DATA_SIZE;
        }else{
            if (m_stMessage.GetLength() - len < APP_DATA_SIZE){//last
            this->mp_UnderLayer->Send((unsigned char*) (LPCTSTR) m_stMessage.Mid(len, m_stMessage.GetLength() - len),
                m_stMessage.GetLength() - len, CHAT_TYPE, CHAT_NOTREC);
            break;
        }
        this->mp_UnderLayer->Send((unsigned char*) (LPCTSTR) m_stMessage.Mid(len, APP_DATA_SIZE), APP_DATA_SIZE, CHAT_TYPE, CHAT_NOTREC);
        len += APP_DATA_SIZE;
        }
    }
}
else{ //lower
    this->mp_UnderLayer->Send((unsigned char*) (LPCTSTR) m_stMessage, m_stMessage.GetLength(), CHAT_TYPE, CHAT_REC);
}
```

2_1 메시지 전송) ChatAppLayer::Send

```
BOOL CChatAppLayer::Send(unsigned char *ppayload, int nlength, int port, unsigned char type)
{
    m_sHeader.app_length = (unsigned short) nlength;
    BOOL bSuccess = FALSE;

    m_sHeader.app_type = type;

    memcpy(m_sHeader.app_data, ppayload, nlength);

    bSuccess = mp_UnderLayer->Send((unsigned char*) &m_sHeader, nlength + APP_HEADER_SIZE, port);

    return bSuccess;
}
```

상위 계층인 IPCAppDlg에서 받아온 데이터에 ChatAppLayer의 header를 붙여주고 자신의 header를 붙여준 Data를 하위 계층인 TCPLayer로 전송을 해준다.

2_2 파일 전송) FileLayer::Send

```
sendLength = m_fileDes.Read(filebuf, FILE_DATA_SIZE);

m_sHeader.fapp_totlen = sendLength;
memcpy(m_sHeader.fapp_data, filebuf, sendLength);

mp_UnderLayer->Send((unsigned char*) &m_sHeader, sendLength + FILE_HEADER_SIZE, FILE_TYPE);
m_sHeader.fapp_seq_num++;
totalLength += sendLength;
((CIPCAppDlg*) mp_aUpperLayer[0])->m_progress.SetPos(totalLength);

//if Packet is lastPacket, send one more packet
if (fileInfo.totalLength - totalLength <= FILE_DATA_SIZE) {
    m_sHeader.fapp_type = DATA_LAST;//change type to DATA_LAST

    sendLength = m_fileDes.Read(filebuf, (fileInfo.totalLength - totalLength));
    m_sHeader.fapp_totlen = sendLength;

    memcpy(m_sHeader.fapp_data, filebuf, sendLength);
    totalLength += sendLength;


    CString message;
    message.Format("Count >> %d", m_sHeader.fapp_seq_num);//sended packet's num
    int index = ((CIPCAppDlg*) mp_aUpperLayer[0])->m_ListChat.AddString(message);
    ((CIPCAppDlg*) mp_aUpperLayer[0])->m_ListChat.SetCurSel(index);

    mp_UnderLayer->Send((unsigned char*) &m_sHeader, sendLength + FILE_HEADER_SIZE, FILE_TYPE);
    //change progressbar
    ((CIPCAppDlg*) mp_aUpperLayer[0])->m_progress.SetPos((int) m_fileDes.GetLength());
```

```
else {//file size smaller than FILE_MAX_SIZE
    m_sHeader.fapp_type = DATA_LAST;
    m_sHeader.fapp_totlen = m_fileDes.GetLength();
    m_sHeader.fapp_msg_type = MSG_DATA;

    sendLength = m_fileDes.Read(filebuf, m_sHeader.fapp_totlen);

    memcpy(m_sHeader.fapp_data, filebuf, sendLength);
    mp_UnderLayer->Send((unsigned char*) &m_sHeader, sendLength + FILE_HEADER_SIZE, FILE_TYPE);
    ((CIPCAppDlg*) mp_aUpperLayer[0])->m_progress.SetPos((unsigned int) m_fileDes.GetLength());
}
```

send함수에서는 상위 계층에서 받아온 파일의 경로와 길이 포트 정보를 이용을 하여 파일을 전송을 한다. 처음에는 먼저 파일명을 얻고 버퍼에 저장을 하고. FileLayer의 header field들과 프로그레스 바를 설정을 해준다. 맨 처음은 파일에 대한 정보들 만이 담긴 Packet을 전송을 하여 받은 쪽에서는 전체 크기를 확인하면서 단편화를 할지 하지 않을지 결정을 할 수 있다. 전송을 한 사람은 return Packet이 올 때까지 while문을 통해서 대기를 하면서 .return Packet이 오면 파일의 크기가 FILE_DATA_SIZE 보다 크다면 Packet을 나누어서 전송을 하고 프로그레스 바를 갱신을 해준다. 마지막 Packet을 전송을 할 때에는 header file의 type값에 마지막 Packet이라는 정보를 주어준다. Packet이 나누어서 보낼 필요가 없을 정도의 크기라면 header file의 type값을 마지막 Packet이라고 설정을 해주고 전송을 해 준다.

(그림 1. 크기가 큰 파일을 나누어서 보내는 부분 /
그림 2. 크기가 작은 파일을 전송을 하는 부분)

3) TCPLayer::Send

```
BOOL CTCPLayer::Send(unsigned char *ppayload, int nlength, int port)
{
    memset(m_sHeader.tcp_data, 0, TCP_DATA_SIZE);
    memcpy(m_sHeader.tcp_data, ppayload, nlength);
    BOOL bSuccess = FALSE;

    m_sHeader.tcp_type = port;

    bSuccess = mp_UnderLayer->Send((unsigned char*) &m_sHeader, nlength + TCP_HEADER_SIZE);

    return bSuccess;
}
```

상위 계층인 ChatAppLayer에서 받아온 데이터를 TCPLayer의 header에 추가하고, m_sHeader.tcp.type필드를 전달받은 port값으로 초기화 시켜준다. 이 port값은 StdAfx.h에 정의되어 있는 값으로 ChatAppLayer를 통해 전달된 메시지 패킷의 경우 0x2080, FileLayer를 통해 전달된 파일 패킷인 경우 0x2090을 가진다. 패킷전송 이후 TCPLayer의 Receive에서 이 필드를 다시 살펴보고 0x2080이면 메시지가 전송되어 왔으므로 ChatAppLayer로 보내고, 0x2090이면 파일이 전송되어 왔으므로 FileLayer으로 보낸다. 이 후 하위 계층인 IPLayer로 TCPLayer의 헤더를 포함한 데이터를 전송을 한다.

4) IPLayer::Send

```
BOOL CIPLayer::Send(unsigned char *ppayload, int nlength)
{
    memset(m_sHeader.ip_data, 0, IP_DATA_SIZE);
    memcpy(m_sHeader.ip_data, ppayload, nlength);
    BOOL bSuccess = FALSE;

    bSuccess = mp_UnderLayer->Send((unsigned char*) &m_sHeader, nlength + IP_HEADER_SIZE);

    return bSuccess;
}
```

상위 계층인 TCPLayer에서 받아온 데이터에 IPLayer의 header를 붙여주고 자신의 header를 붙여준 Data를 하위 계층인 EthernetLayer로 전송을 해준다.

5) EthernetLayer::Send

```
BOOL CEthernetLayer::Send(unsigned char *ppayload, int nlength)
{
    memset(m_sHeader.enet_data, 0, ETHER_MAX_DATA_SIZE);
    memcpy(m_sHeader.enet_data, ppayload, nlength);
    BOOL bSuccess = FALSE;

    bSuccess = mp_UnderLayer->Send((unsigned char*) &m_sHeader, nlength + ETHER_HEADER_SIZE);

    return bSuccess;
}
```

상위 계층인 IPLayer에서 받아온 data를 자신에 대한 정보인 EthernetLayer의 header에 붙여주고, 하위 계층인 NILayer의 Send함수를 호출을 하여 송신과정을 진행한다.

6) NILayer::Send

```
BOOL CNILayer::Send(unsigned char* ppayload, int nlength)
{
    OID_GEN_MEDIA_CONNECT_STATUS;
    //pcap_sendpacket(send packet)
    if(pcap_sendpacket(m_pAdapterObjects, ppayload, nlength) != 0)
    {
        fprintf(stderr, "\nError sending the packet: \n", pcap_geterr(m_pAdapterObjects));
        return FALSE;
    }

    return TRUE;
}
```

상위 계층인 EthernetLayer에서 온 data를 WinPcap의 내부 함수인 pcap_sendpacket() 함수를 이용을 하여 네트워크로 연결이 되어진 다른 PC에 전송을 하고 전송이 성공 하였다면 sendpacket()함수는 0을 반환해서 최종적으로 TRUE를 반환을 하고 실패를 하였다면 -1을 반환해서 if문으로 들어가 오류를 출력을 하고 FALSE를 반환을 한다.

나. 수신

1) IPCAppDlg::ReceiveThread

```
UINT CIPCAppDlg::ReceiveThread(LPVOID pParam)
{
    CIPCAppDlg* obj = (CIPCAppDlg*) pParam;

    //if m_bSendReady is on, start Thread
    while (obj->m_bSendReady == TRUE)
    {
        CBaseLayer* bLayer;
        bLayer = obj->m_LayerMgr.GetLayer("ChatApp");
        unsigned char *ppayload = bLayer->Receive(0);
        if (ppayload != NULL)
            obj->Receive(ppayload);
    }
    return 0;
}
```

ReceiveThread()함수에서는 while문을 돌면서 입력이 올 때까지 기다리다가 입력이 주어지면 ChatAppLayer계층의 객체를 이용하여서 ChatAppLayer의 Receive()함수를 호출을 하고 반환이 되어진 데이터의 값을 출력을 한다.

2_1 메시지 수신) ChatAppLayer::Receive

```
unsigned char* CChatAppLayer::Receive(int Thread_type)//Chat
{
    unsigned char* ppayload = mp_UnderLayer->Receive(Thread_type);

    if (ppayload != NULL){
        PCHAT_APP_HEADER app_hdr = (PCHAT_APP_HEADER) ppayload;
        int length = app_hdr->app_length;


        memset(chatbuf, '₩0', APP_DATA_SIZE + 2);
        memcpy(chatbuf, app_hdr->app_data, length);

        if (app_hdr->app_type == CHAT_REC){//REC type
            chatbuf[APP_DATA_SIZE + 1] = 1;//masking 1
            return (unsigned char*) chatbuf;
        }
        else if (app_hdr->app_type == CHAT_NOTREC){//NOTREC type
            chatbuf[APP_DATA_SIZE + 1] = 0;//masking 0
            return (unsigned char*) chatbuf;
        }
    }
    return 0;

}
```

하위 계층인 TCPLayer의 Receive를 호출을 하여 얻은 반환값의 데이터에서 자신의 header에 있는 데이터를 검사 한 후 상위 계층인 IPCAppDlg에 반환을 해 준다.

2_2 파일 수신) FileLayer::Receive

```
else if (BooleanErr  && fileHead->fapp_type == DATA_MAIN) {
    //error process
    //send sucess packet seq_num to Sender

    if ((check + 1 != fileHead->fapp_seq_num) && (fileHead->fapp_seq_num != 0))
    {
            //send just one ERR packet to Sender
        if (BooleanErr){
            fileApp.fapp_msg_type = MSG_ERR;//fapp_msg_type == ERR
            fileApp.fapp_seq_num = check+1;
            mp_UnderLayer->Send((unsigned char*) &fileApp, FILE_HEADER_SIZE, FILE_TYPE);
            BooleanErr = FALSE;
        }
    }
    else
    {
        ((CIPCAppDlg*) mp_aUpperLayer[0])->m_progress.SetPos(check * fileHead->fapp_totlen);
        check = fileHead->fapp_seq_num;
        m_recvFile.Write(fileHead->fapp_data, fileHead->fapp_totlen);
    }
}
else if (BooleanErr && fileHead->fapp_type == DATA_LAST){//fapp_type == DATA_LAST

    fileApp.fapp_msg_type = MSG_LASTOK;
    mp_UnderLayer->Send((unsigned char*) &fileApp, FILE_HEADER_SIZE, FILE_TYPE);

    m_recvFile.Write(fileHead->fapp_data, fileHead->fapp_totlen);
    m_recvFile.Close();
    AfxMessageBox("파일전송을 완료했습니다");
}
```

Receive()함수는 Thread로 인해 계속 호출이 된다. 아래 계층에서 올라오는 Packet가 없으면 함수는 아무 일도 하지 않지만, 하위 계층으로부터 Packet이 오게 되면 Packet이 첫 번째, 중간, 마지막인가에 따라서 다르게 처리를 한다. Packet이 첫 번째 부분이라면 그 파일 정보를 받아서 파일을 생성을 하고 생성 시 파일명으로 파일 크기만큼 공간을 확보를 한다. 중간 Packet이라면 생성된 파일에 데이터를 덧붙이는 과정을 거친다. 전달받은 Packet 마지막 부분이라면 생성된 파일의 크기 및 순서를 고려, 잘 전송 받았는지 확인을 하고 파일을 닫고 파일 수신 과정을 종료를 한다.

3) TCPLayer::Receive

```
unsigned char* CTCPLayer::Receive(int Thread_type)
{
    static PTCP_HEADER pFrame;
    if (Thread_type == 2){// type 2
        unsigned char* ppayload = mp_UnderLayer->Receive();
        if (ppayload != NULL){
            pFrame = (PTCP_HEADER) ppayload;
            if ((pFrame->tcp_type == CHAT_TYPE || pFrame->tcp_type == FILE_TYPE)){
                _Rock = TRUE;
                while (_Rock);
            }
        }
    }
    else if (_Rock){//type 0 , 1
        if (pFrame->tcp_type == CHAT_TYPE && !Thread_type){//return to ChatAppLayer
            memcpy(chat_Packet, (unsigned char*) pFrame, IP_DATA_SIZE);//C
            PTCP_HEADER C_pFrame = (PTCP_HEADER) chat_Packet;
            _Rock = FALSE;
            return C_pFrame->tcp_data;//bSuccess = mp_aUpperLayer[0]->Receive();
        }
    }
    return 0;
}
```

Thread의 type에 따라서 파일이 아닐 경우 하위 계층인 IPLayer의 Receive를 호출을 하여 얻은 반환값 데이터를 상위 계층인 IPCAppDlg에 반환을 해 준다. 하나의 Thread_type이 2인 Thread가 하위계층으로부터 Receive를 통해 데이터를 받아오면 _Rock가 True로 바뀌게된다. 이 순간 다음 if문으로 들어가 데이터의 타입이 File인지 chat인지 구분하고 File이면 FileLayer에 chat이면 ChatAppLayer로 프레임의 데이터부분을 반환해준다.

4) IPLayer::Receive

```
unsigned char* CIPLayer::Receive()
{
    unsigned char* ppayload = mp_UnderLayer->Receive();

    if (ppayload != NULL){
        PIP_HEADER pFrame = (PIP_HEADER) ppayload;

        return pFrame->ip_data;
    }
    return 0;
}
```

하위 계층인 EthernetLayer의 Receive()함수를 호출을 하여 얻은 반환 값을 이용을 하여서 만약 하위 계층에서 온 값이 NULL이 아니라면 상위 계층에 자신의 header를 분리한 데이터를 전송을 해준다.

5) EthernetLayer::Receive

```
unsigned char* CEthernetLayer::Receive()
{
    unsigned char* ppayload = mp_UnderLayer->Receive();

    unsigned char   ed[6];
    unsigned char   es[6];
    memset(ed, 0, 6);
    memset(es, 0, 6);

    if (ppayload != NULL){
        PETHERNET_HEADER pFrame = (PETHERNET_HEADER) ppayload;
        unsigned char F = 255;// = 0xff(breadcast)
        memcpy(ed, pFrame->enet_dstaddr, 6);
        memcpy(es, pFrame->enet_srcaddr, 6);

        int i, check = 0;
        for (i = 0; i < 6; i++)  //Comapre packet's adress and My MacAdress
        if ((ed[i] != m_sHeader.enet_srcaddr[i]) && (ed[i] != F|| (es[i] == m_sHeader.enet_srcaddr[i]))){//des and MacAdress comapre + des and
            check = 1;
            break;
        }
        if (!check){  //is sucess to comapre and go upper layer
            return pFrame->enet_data;
        }
    }
    return 0;
}
```

하위 계층인 NILayer의 Receive()를 호출을 하여 Data를 반환을 하여 가져온다. 그 후 나의 Mac주소와 Packet의 주소를 비교를 하여 같으면 상위계층으로 전송을 하고 다르면 0을 반환을 한다.

6) NILayer::Receive

```
unsigned char* CNILayer::Receive()
{
    int result;
    struct pcap_pkthdr *header;
    unsigned char data[ETHER_MAX_SIZE];
    const unsigned char* pkt_data;
    //make packet's memory
    memset(data, '₩0', ETHER_MAX_SIZE);
    //pcap_next_ex(receive packet)
    while((result = pcap_next_ex( m_pAdapterObjects, &header, &pkt_data)) >= 0) {
        if(result == 0)
            continue;

        memcpy(data, pkt_data, ETHER_MAX_SIZE);//copy packet
        return data;//return packet
    }

    if(result = -1) {//error
        printf("Error reading the packets : %s₩n", pcap_geterr(m_pAdapterObjects));
        return 0;
    }

    return 0;
}
```

WinPcap의 내장함수인 pcap_next_ex()를 이용을하여 다음 packet이 있으면 상위계층으로 전송을 할 데이터에 복사를 하여 반환을 해주고 패킷을 읽는데 에러가 발생하면 에러문을 출력을 하고 0을 반환을 한다.

다. Packet32.h를 이용하여 MAC정보 얻기

1) NILayer.h에서 packet32.h를 포함시킨다. `#include "packet32.h"`

2) Dialog가 팝업되기 전 NILayer의 SetNICList() 메소드 실행

가) SetNICList()에서 pcap_h의 메소드인 pcap_findalldevs(&m_allDevs, errbuf)를 실행

```
if(pcap_findalldevs(&m_allDevs, errbuf) == -1){
    fprintf(stderr, "Error in pcap_findalldevs: %s₩n", errbuf);
    exit(1);
}
```

NILayer.h에 선언되어있는 m_allDevs의 타입은 pcap_if_t*로 pcap_h에 선언되어 있다.

```
struct pcap_if{
        struct pcap_if *next; // 다음 인터페이스의 노드를 가리키는 포인터
        char *name; // NIC리스트에 뜨는 디바이스의 이름
        char *description; // 인터페이스의 이름
        struct pcap_addr *addresses; // MAC 주소값
};
```

(1) 모든 인터페이스를 순회하며 정보를 m_allDevs에 저장한다.
(2) 첫 번째 인터페이스의 정보는 m_allDevs에 저장되어 있다.
(3) 다음 인터페이스의 정보는 next 노드로 연결하여 리스트를 유지한다.
(4) 마지막 노드의 next는 NULL값을 가진다.

나) 위에서 저장한 인터페이스 정보를 이용하여 어댑터의 수를 필드에 저장

```
for(d = m_allDevs; d; d=d->next)
    m_iNumAdapter++;
```

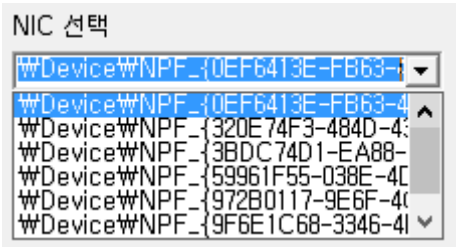
3) IPCAppDlg클래스의 OnInitDialog()에서 NIC리스트를 설정

```
for (int i = 0; i < m_NI->getAdapterNum(); i++){
    m_NICSet.AddString(m_NI->getAdapterName(i));
}
```

가) getAdapterNum()을 통해 어댑터의 개수 반환하여 반복문의 index로 사용

나) getAdapterName()을 호출하여 m_allDevs의 name에 접근하여 디바이스이름을 리턴한다. ex) "₩₩Device₩₩NPF_{0EF6413E-FB63-4A9D-9EB4-CF94F1B34141}"

4) Main dialog가 실행되면 다음과 같이 드랍다운 리스트가 표현된다.



라. 얻은 MAC정보를 이용하여 NIC 선택

1) NIC리스트 선택, 재설정 버튼 클릭 시 IPCAppDlg의 OnButtonAddrSet() 실행

가) 선택한 Adapter의 index를 필드에 저장하는 setAdapterNum() 호출

```
m_NI->setAdapterNum(m_NICSet.GetCurSel());
```

(1) Adapter의 index를 저장한다. `m_selectedNum = index;`

(2) Adapter의 name을 저장한다. `getAdapterName(index);`

(3) 저장한 Adapter의 name을 이용하여 pcap_open_live메소드를 호출한다. pcap_open_live로 첫 번째 인자로 주어지는 network device에 대한 packet capture descriptor(PCD)를 만든다. 패킷을 capture하는 일은 대부분 PCD를 이용해서 이루어진다.

나) setMacAddress() 호출하여 선택한 Adapter의 Mac주소 NILayer의 필드에 저장

```
m_NI->setMacAddress();
```

다) getMacAddress() 호출하여 NILayer에 설정한 Mac주소를 m_SrcMacAddress에 저장

```
m_SrcMacAddress = m_NI->getMacAddress();
```

라) m_SrcMacAddress에 저장한 Mac주소의 Hex Format을 변형

```
m_stSrcAdd.Format("%.2X-%.2X-%.2X-%.2X-%.2X-%.2X",
    m_SrcMacAddress[0], m_SrcMacAddress[1], m_SrcMacAddress[2],
    m_SrcMacAddress[3], m_SrcMacAddress[4], m_SrcMacAddress[5]);
```

마) 이더넷레이어의 필드에 소스와 데스티네이션 주소를 설정

```
m_Ether->SetSourceAddress(m_SrcMacAddress);
m_Ether->SetDestinAddress(m_DstMacAddress);
```

바) 3개의 스레드를 시작하여 packet을 지속적으로 검사

```
m_RecvThread1 = ::AfxBeginThread(CIPCAppDlg::ReceiveThread, this);
m_RecvThread2 = ::AfxBeginThread(CIPCAppDlg::ReceiveFileThread, this);
m_RecvThread3 = ::AfxBeginThread(CIPCAppDlg::ReceiveTCPThread, this);
```

마. TCPLayer에서 사용하는 Port번호

StdAfx.h 헤더파일에 선언하였다.

```
#define CHAT_TYPE               0x2080
#define FILE_TYPE               0x2090
```

1) Dest_Port가 0x2080이면 ChatAppLayer에게 데이터를 넘긴다.

2) Dest_Port가 0x2090이면 FileAppLayer에게 데이터를 넘긴다.

바. Thread 사용

1) ChapAppLayer 계층 – 전달된 문자 화면에 출력

```
UINT CIPCAppDlg::ReceiveThread(LPVOID pParam)
{
    CIPCAppDlg* obj = (CIPCAppDlg*) pParam;

    //if m_bSendReady is on, start Thread
    while (obj->m_bSendReady == TRUE)
    {
        CBaseLayer* bLayer;
        bLayer = obj->m_LayerMgr.GetLayer("ChatApp");
        unsigned char *ppayload = bLayer->Receive(0);
        if (ppayload != NULL)
            obj->Receive(ppayload);
    }
    return 0;
}
```

MAC 주소를 설정을 하게되면 Thread들이 실행이 되어진다. Thread가 실행이 되어지는 계층 중 하나는 ChatAppLayer이다 ChatAppLayer에서는 문자를 계속 받기 위해서 Thread를 사용을 하는데, while문에서 m_bSendReady가 True인 동안 계속 while문을 반복을 하면서 ChatAppLayer의 receive()함수를 호출을 한다. 하위 계층으로 만약 데이터를 받으면 payload의 값이 NULL이 아니다. 그러면 IPCAppDlg 클래스에 Receive함수를 호출을 하여 화면에 전달된 값을 출력을 하고 하위 계층으로부터 데이터를 받지 못하였다면 계속 반복문을 수행을 한다.

2) FileLayer 계층 – 파일 보내기

```
UINT CIPCAppDlg::FileThread(LPVOID pParam)
{
    CIPCAppDlg* obj = (CIPCAppDlg*) pParam;

    obj->m_File->Send((unsigned char*) obj->m_strFile.GetBuffer(0), obj->m_strFile.GetLength(), FILE_TYPE);

    obj->FileSended=FALSE;//init
    obj->TransferdButtonChange();
    return 0;
}
```

이 FileThread()함수는 파일을 전송을 하면서 다른 일을 하기 위해서 만든 Thread 이다. UI에 있는 전송 버튼을 누르면 이 Thread가 호출이 되어 사용이 된다. 이 쓰레드는 FileLayer에 Send()함수를 호출을 하여 File을 전송을 하며 이 Thread를 이용을 하면 문자를 입력하고 전송 받으면서 파일을 전송을 할 수 있게 된다.

3) FileLayer 계층 – 파일 받기

```
UINT CIPCAppDlg::ReceiveFileThread(LPVOID pParam)
{
    CIPCAppDlg* obj = (CIPCAppDlg*) pParam;

    //if m_bSendReady is on, start Thread
    while (obj->m_bSendReady == TRUE)
    {
        CBaseLayer* bLayer;
        bLayer = obj->m_LayerMgr.GetLayer("File");
        bLayer->Receive(1);//Thread num
    }
    return 0;
}
```

ReceiveFileThread() 함수는 다른 작업을 하면서 File을 받기 위해서 FileLayer 계층의 Receive()함수를 Thread를 이용을 하여 호출을 한다. 맨 처음 MAC 주소가 설정이 되면 Thread가 호출이 된다. FileLayer 계층의 Receive() 함수에서는 계속 호출이 되어지면서 하위 계층으로부터 전송이 되어진 File이 있으면 그 파일을 자신이 설정을 한 경로에 저장을 한다. 만약 하위 계층으로부터의 데이터가 없다면 Receive()함수에서는 아무 동작을 하지 않는다.

4) TCPLayer 계층 – 전달된 패킷 받기

```
UINT CIPCAppDlg::ReceiveTCPThread(LPVOID pParam)
{
    CIPCAppDlg* obj = (CIPCAppDlg*) pParam;

    //if m_bSendReady is on, start Thread
    while (obj->m_bSendReady == TRUE)
    {
        CBaseLayer* bLayer;
        bLayer = obj->m_LayerMgr.GetLayer("Tcp");
        bLayer->Receive(2);//Thread num
    }

    return 0;
}
```

실습 강의 자료에서는 NILayer에서 Thread를 사용을 한다. 하지만 우리 조는 TCPLayer에서 Thread를 구현을 하였다. 우리 조는 상위 Layer에서 하위 Layer의 Receive()함수를 호출을 하여 전달이 되어진 패킷을 받아오는 형식으로 구성을 하였다. ReceiveTCPThread() 함수 또한 앞서 설명한 FileReceiveThread()와 ReceiveThread()처럼 Mac 주소가 설정이 되어지면 Thread가 실행이 되어진다. while문을 계속 반복을 하면서 TCPLayer의 Receive()함수를 호출을 하게된다. TCPLayer의 Receive()에서는 계속 하위 계층으로부터 패킷을 받는다. 패킷을 받게 되면 TCPLayer의 Receive함수는 다른 계층에서 깨우기 전까지는 계속 while문을 돌리며 대기를 하고 패킷을 받지 못하면 아무 일도 하지 않는다.

사. File 전송

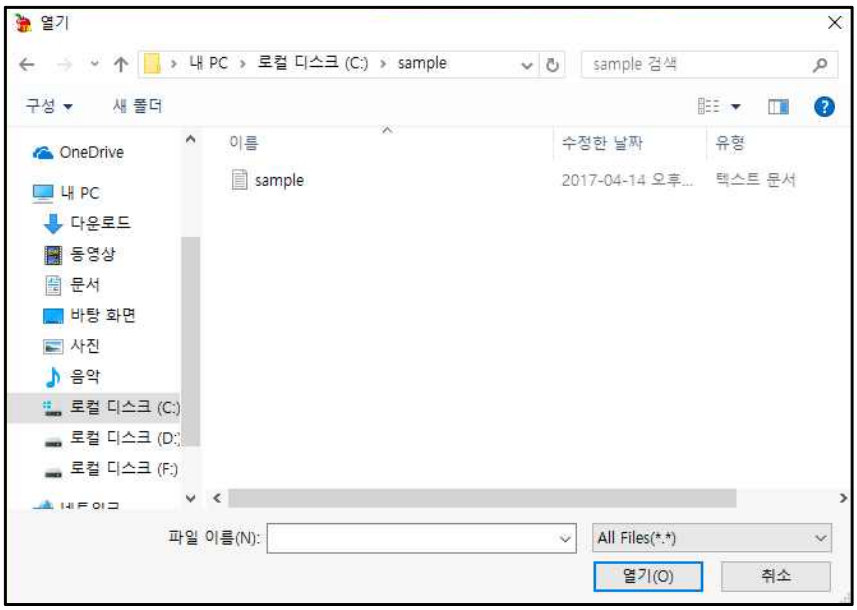
1) Main dialog에서 "파일찾기" button 클릭 시 OnButtonFileserch() 호출

```
void CIPCAppDlg::OnButtonFileserch()
{ 경과 시간 21,541ms 이하
    CFileDialog fd(TRUE, NULL, NULL, OFN_HIDEREADONLY, "All Files(*.*)|*.*||");

    if (IDOK == fd.DoModal()) {
        m_strFile = fd.GetPathName();
    }
    UpdateData(FALSE);
}
```

파일열기 대화상자 클래스 CFileDialog를 이용한다.

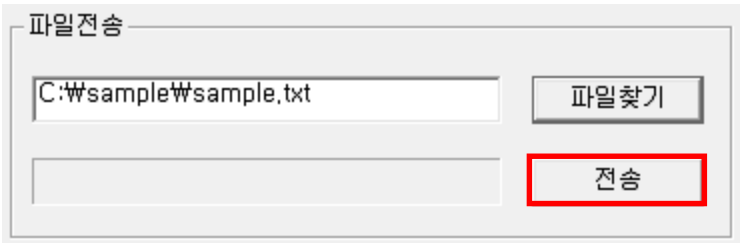
가) "파일찾기" button클릭 시, DoModal()이 호출되어 dialog가 팝업된다.



나) 전송할 파일을 정상적으로 선택하였다면 m_strFile에 파일의 경로명을 저장한다.
ex) m_strFile = "C:₩sample₩sample.txt"

다) UpdateData(FALSE);로 변수의 값을 화면에 띄워준다.

2) "전송" button 클릭 시 OnButtonFiletrans() 호출



가) 파일경로명이 전달되었는지 확인하여 `if (!m_strFile.IsEmpty())`,
아직 파일을 선택하지 않았다면 파일을 선택하도록 팝업창을 띄운다.

```
AfxMessageBox("Select a file to send first!");
```

나) 파일이 선택된 상태라면 FileThread를 실행한다.

```
CWinThread* m_FileThread = ::AfxBeginThread(CIPCAppDlg::FileThread, this);
```

아. 각 Layer에서 사용된 프로토콜 header의 field

1) IPCAppDIg.h의 field

가) CWinThread* m_RecvThread1~3 : 각 Thread의 이름
나) CProgressCtrl m_progress : 프로그래스 바의 상태
다) CComboBox m_NICSet : 네트워크 장비의 리스트
라) CListBox m_ListChat : 대화목록
마) CString m_stMessage : 메세지
바) CString m_stSrcAdd : 시작 주소
사) CString m_stDstAdd : 목적지 주소
아) CString m_strFile : 파일의 이름
자) HICON m_hicon : 아이콘
차) CString payload : 실제 데이터
카) CLayerManager m_LayerMgr : LayerManager의 객체
타) int m_nAckReady : File의 첫 Packet이 도착되었는지를 나타내는 상태
파) BOOL m_bSendReady : 보낼 준비가 되었다면 TRUE아니면 FASLE
하) unsigned char* m_SrcMacAddress : 시작지의 Mac주소를 가리키는 포인터
거) unsigned char m_DstMacAddress : 목적지의 Mac주소
너) CChatAppLayer* m_ChatApp : ChatAppLAyer의 객체
더) CEthernetLayer* m_Ether : EthernetLayer의 객체
러) CNILayer* m_NI : NILayer의 객체

2) ChatAppLayer.h의 header field

가) unsigned short app_length : 데이터의 총 길이
나) unsigned char app_unused : 우선 사용 안함
다) unsigned char app_type : 데이터의 타입이다.(2 = notREC, 1 = REC)
라) unsigned char app_data[APP_DATA_SIZE] : ChatApp의 실제 데이터 부분

3) FileLayer.h의 header field

가) unsigned long fapp_totlen : 파일의 총 길이
나) unsigned short faap_type :파일의 데이터 타입
다) unsigned char fapp_msg_type : 메시지의 종류
라) unsigned char ed : 일단 사용 안함
마) unsigned long : fapp_seq_num : fragmentation 순서
바) unsigned char fapp_data[FILE_DATA_SIZE] : 파일이 저장

4) TCPLayer.h의 header field

가) unsigned char tcp_dstaddr[2] : 목적지 주소
나) unsigned chat tcp_srcaddr[2] : 시작 주소
다) unsigned short tcp_type : 데이터의 타입
라) unsigned char tcp_data[TCP_DATA_SIZE] : 실제 데이터 부분

5) IPLayer.h의 header field

가) unsigned char ip_dstaddr[4] : 목적지 주소
나) unsigned chat ip_srcaddr[4] : 시작 주소
다) unsigned short ip_type : 데이터의 타입
라) unsigned char ip_data[IP_DATA_SIZE] : 실제 데이터 부분

6) EthernetLayer.h의 header field

가) unsigned char enet_dstaddr[6] : 목적지 주소
나) unsigned chat enet_srcaddr[6] : 시작 주소
다) unsigned short enet_type : 데이터의 타입
라) unsigned char enet_data[ETHER_MAX_DATA_SIZE] : 실제 데이터 부분

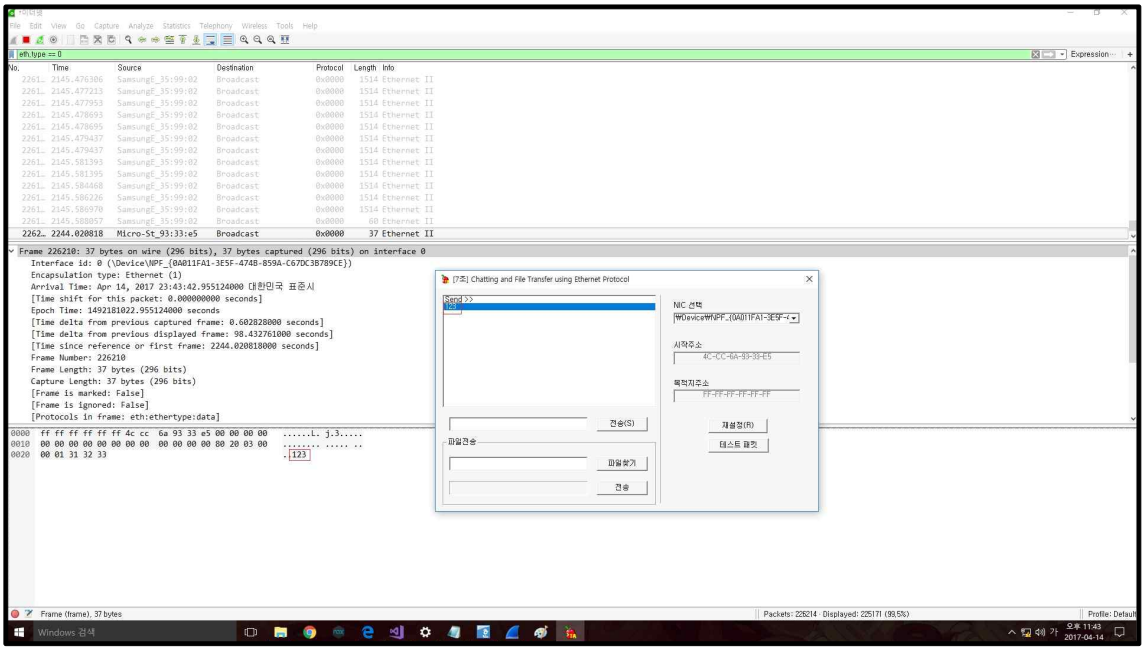
7) NILayer.h의 header field

가) unsigned char m_MacAddress[6] : Mac주소의 값
나) char m_adapterName[1024] : 네트워크 장비 adapter의 이름
다) int m_selectedNum : 선택된 디바이스의 번호
라) int m_iNumAdapter : 열린 네트워크 장비의 수
마) pcap_t* m_pAdaterObjects : Packet을 Capture하기위한 객체
바) pcap_if_t* m_allDevs : 네트워크 디바이스 인터페이스의 리스트 포인터

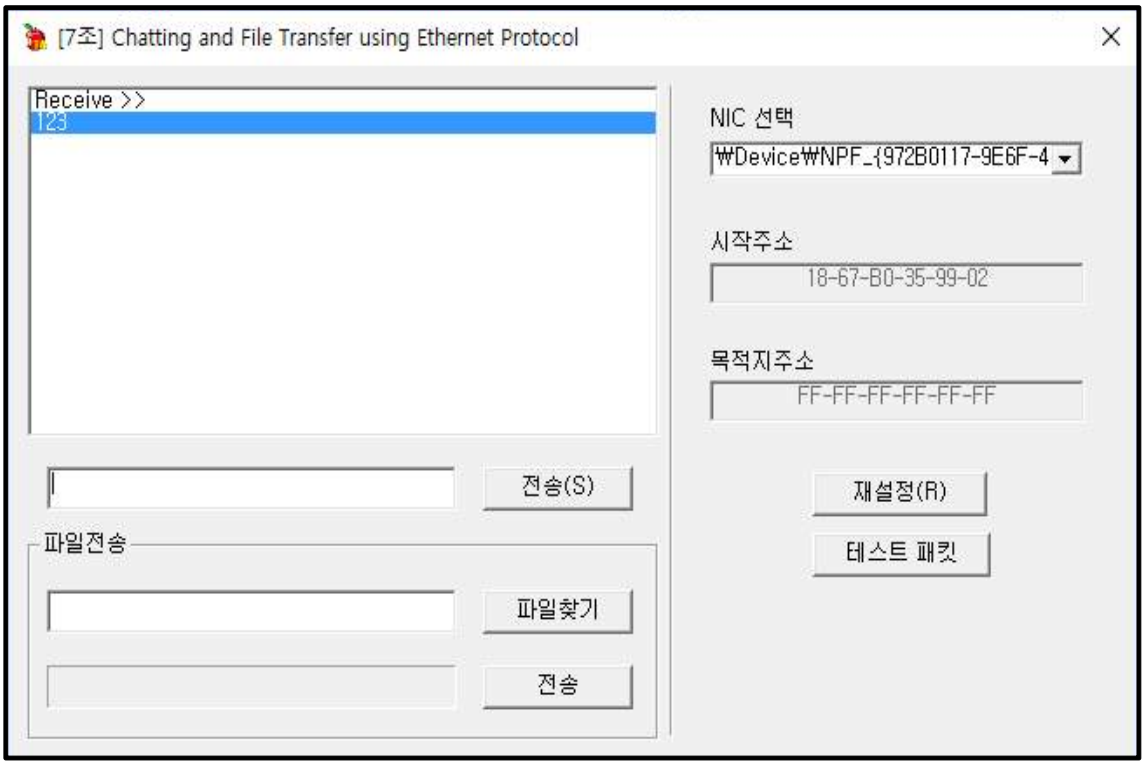
4. 실행 결과

가. 패킷 캡처 화면 및 패킷 분석 설명

1) 송신



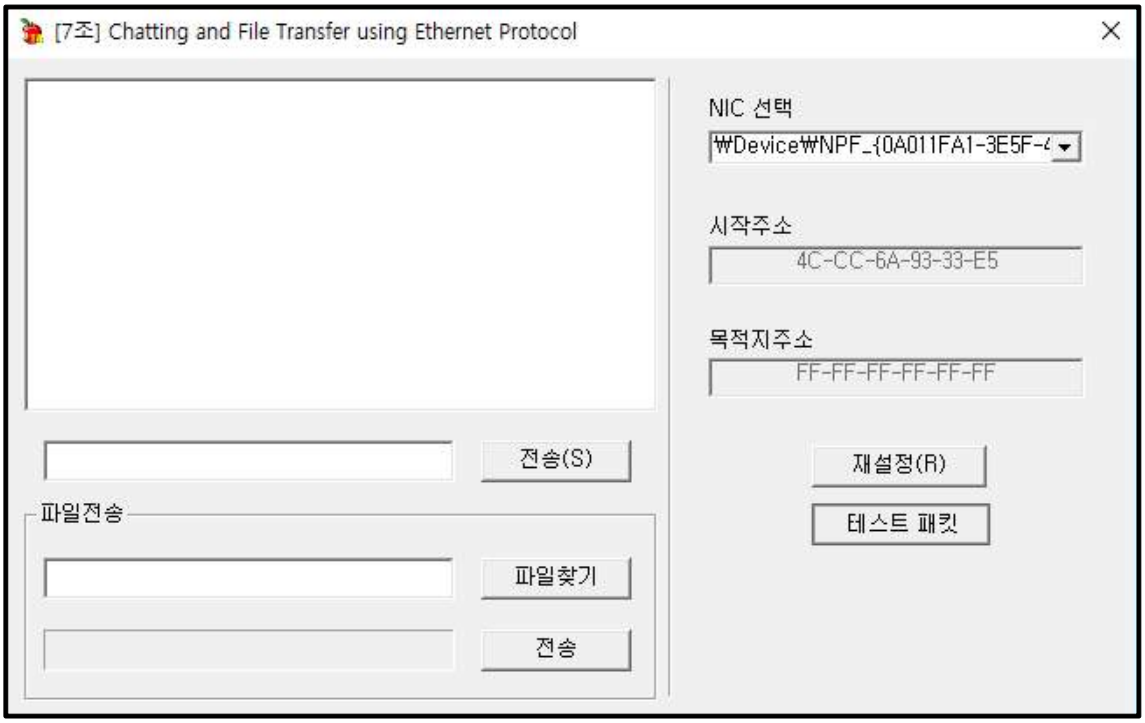
2) 수신



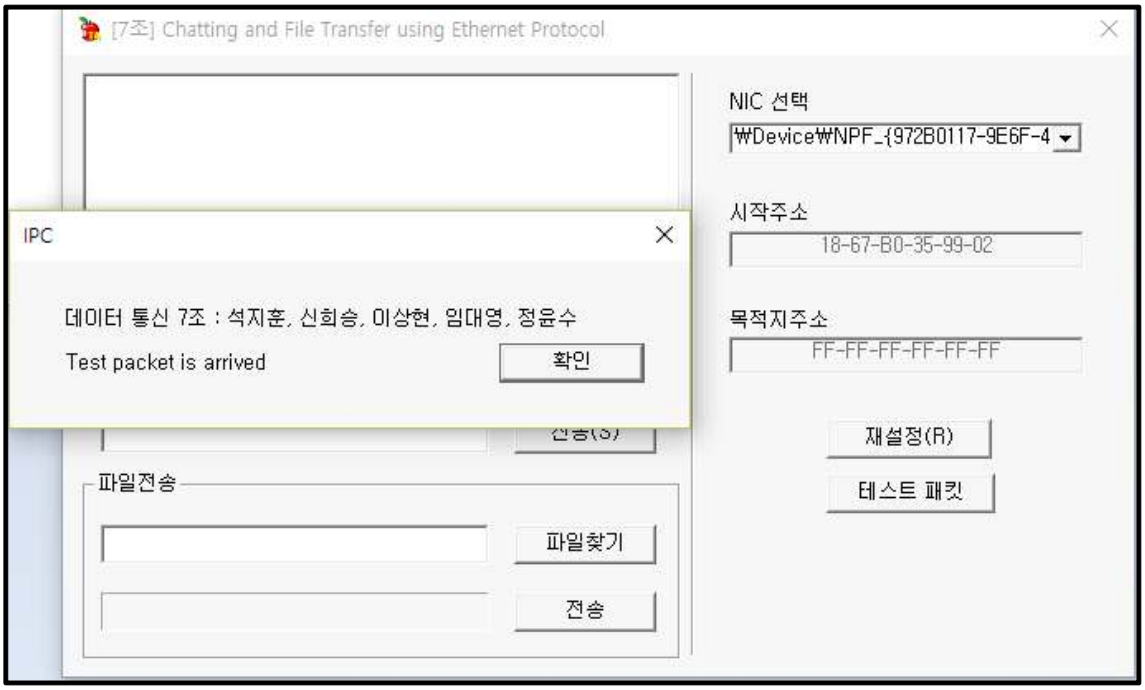
나. 프로그램 결과 화면

1) 테스트 패킷 송신 & 수신

가) 테스트 패킷 송신

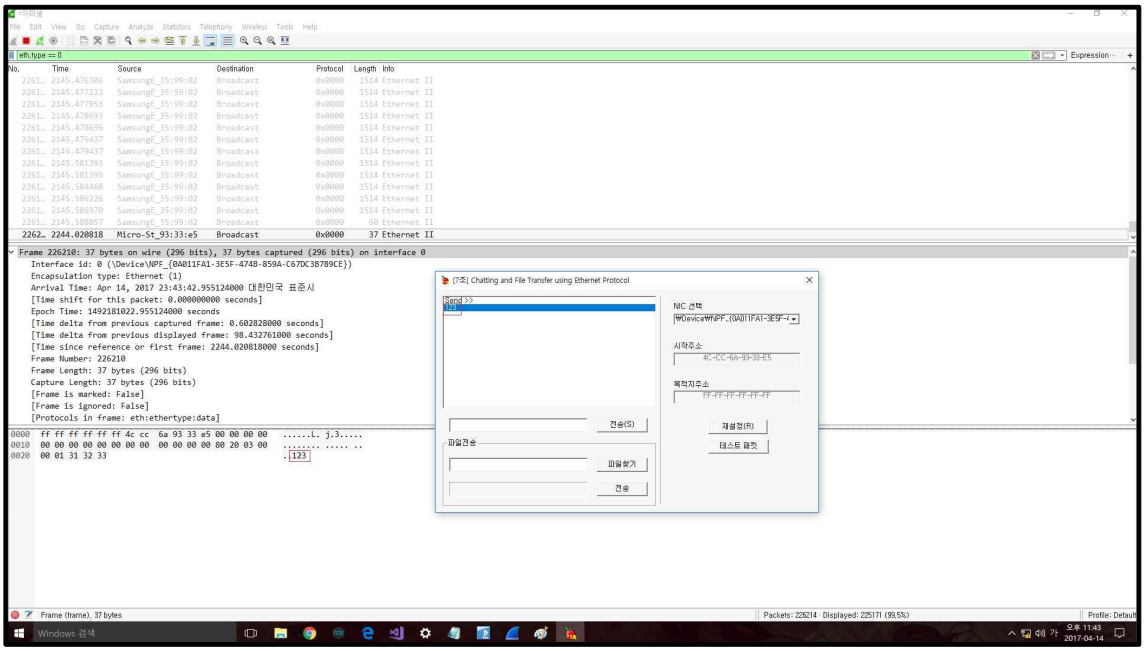


나) 테스트 패킷 수신

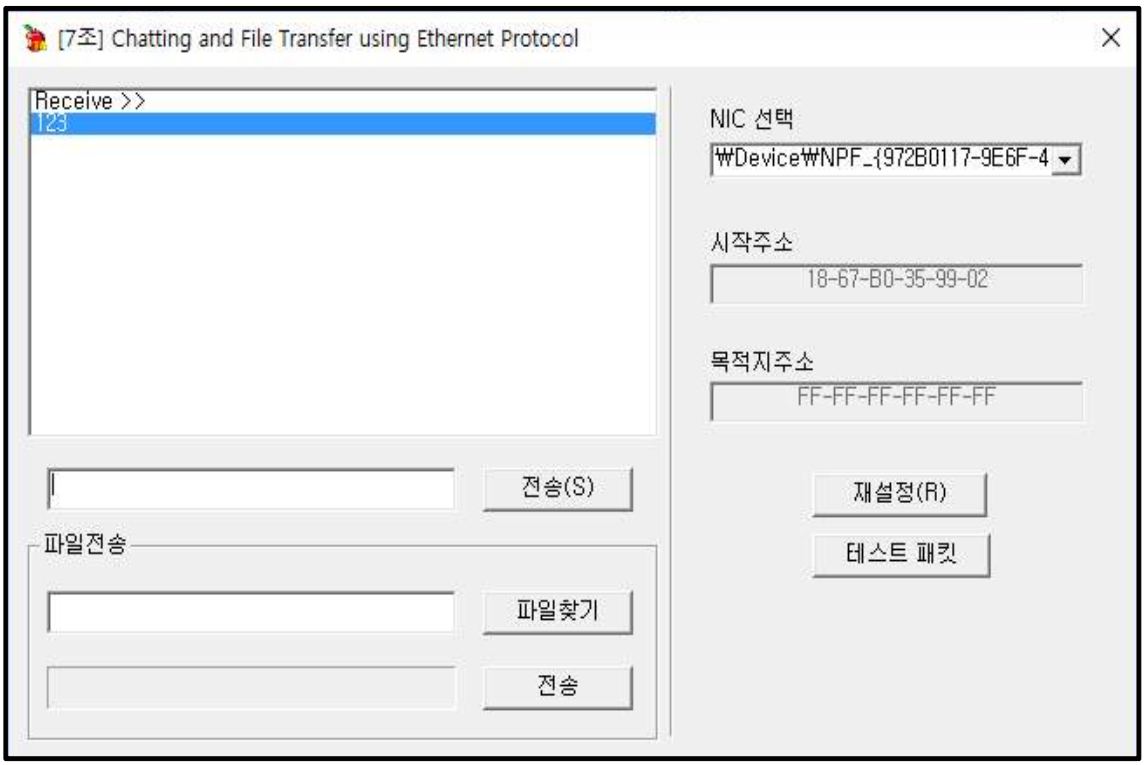


2) 메시지 패킷 송신 & 수신

가) 메시지 패킷 송신



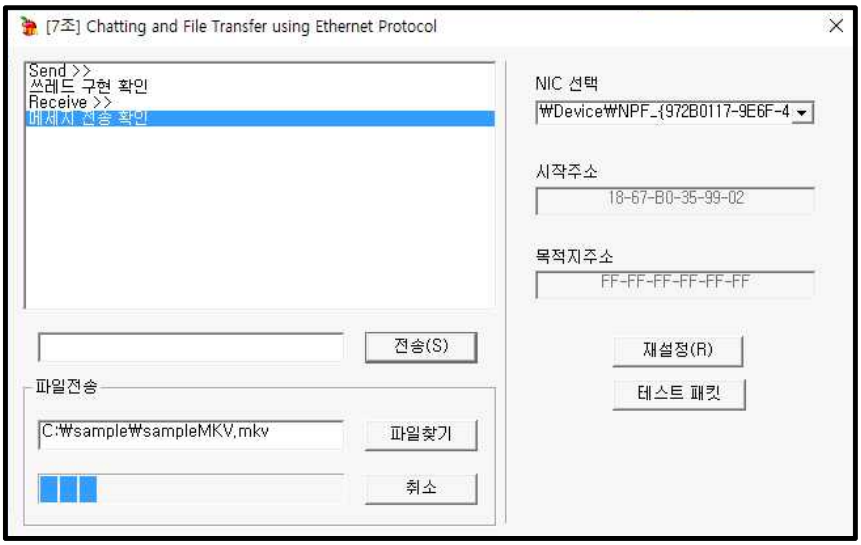
나) 메시지 패킷 수신



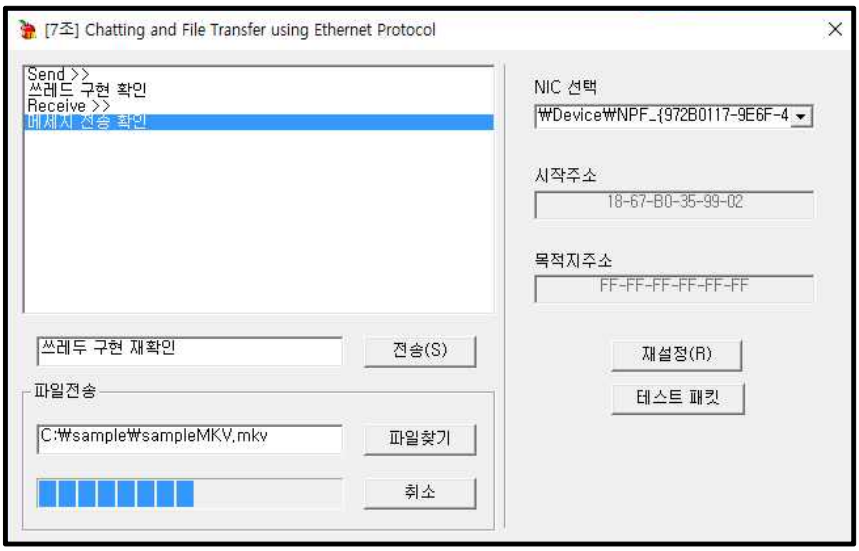
1) 파일 송신 & 수신

가) 파일 패킷 송신

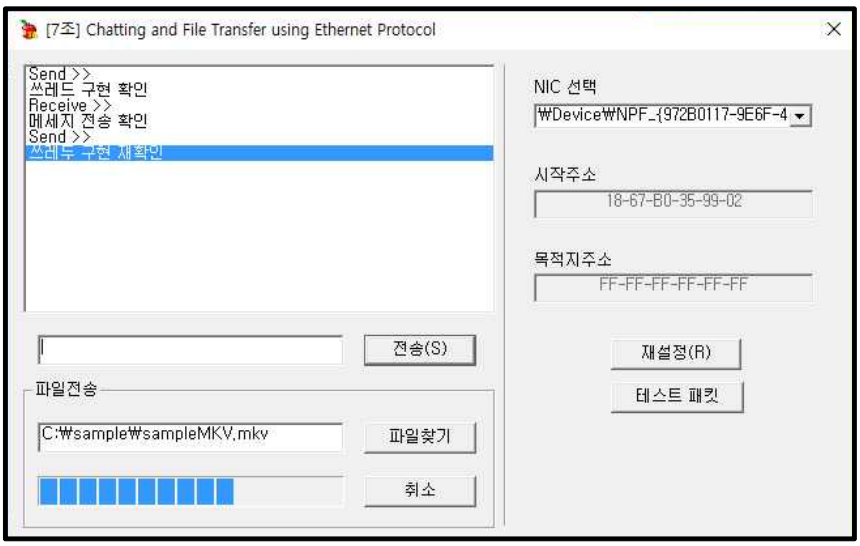
(1) 파일 전송 중



(2) 파일 전송 중 쓰레드를 이용하여 메시지 전송 준비

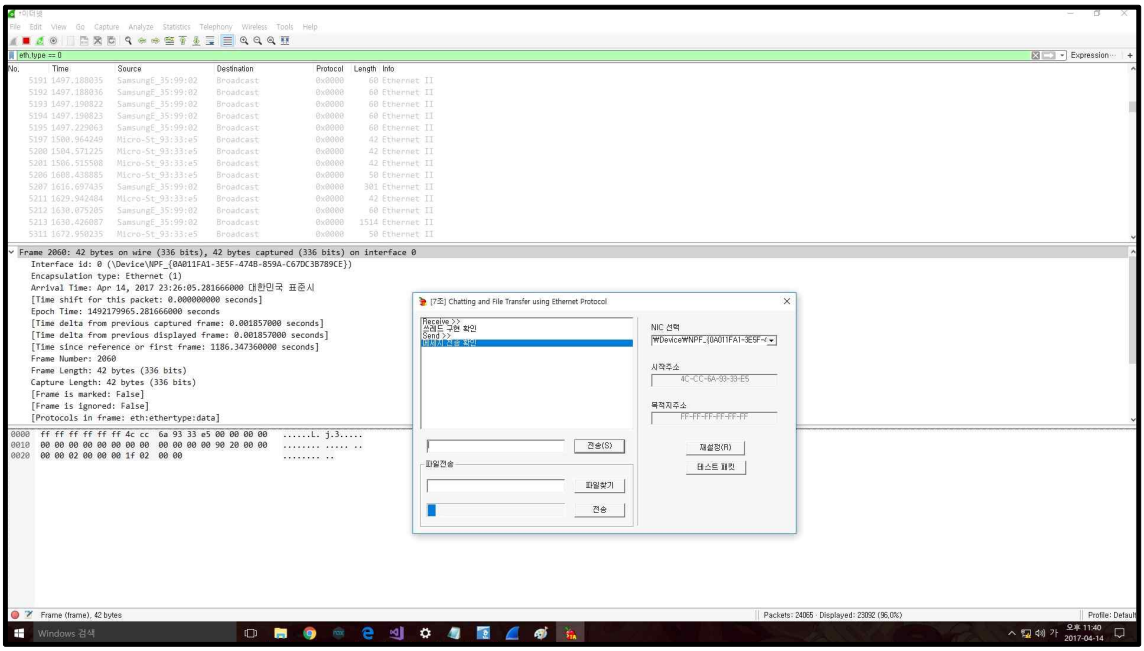


(3) 파일 전송 중 쓰레드를 이용하여 메시지 전송

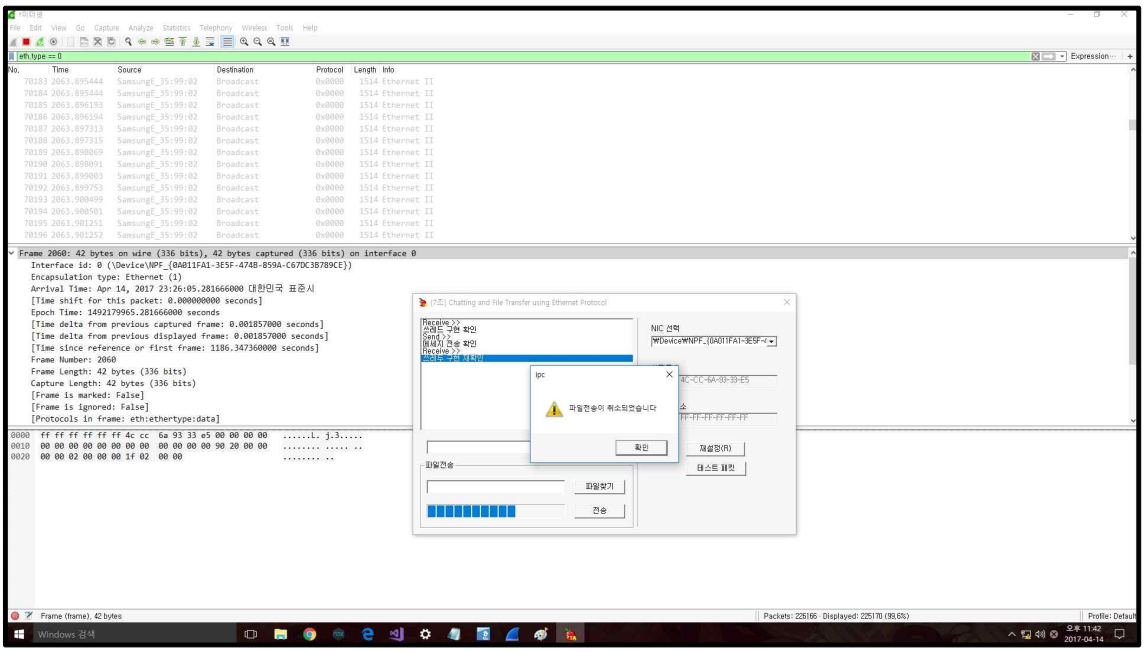


나) 파일 패킷 수신

(1) 파일 전송 중



(2) 파일 전송 중 취소



3) 실습 환경